7N-61-CR

79401

P-1

ESP

Evolution-based Simulation for Placement optimization

(NASA-CR-181041) ESP: EVOLUTION-BASED SIMULATION FOR PLACEMENT OPTIMIZATION (Illinois Univ.) 12 p Avail: NTIS

N87-70458

Unclas
00/61 0079401

Ralph-Michael Kling
Prithviraj Banerjee

Computer Systems Group
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 W. Springfield Av.
Urbana, IL-61801

(217) 333-6564

ABSTRACT

In the past, several heuristic algorithms have been proposed for optimally placing a number of modules such that the total interconnect wire length is minimized. A disadvantage of using those heuristics is their susceptibility to getting stuck at locally optimal solutions. Recently, the simulated annealing (SA) technique has been shown to be immensely successful in achieving globally near-optimal solutions. The main difficulty of any SA algorithm is the tremendous computation time required to solve it. In this paper, we propose a new heuristic algorithm for placing modules. It is based on the analogy of finding the optimally suited modification of an initial system in VLSI-design and in biological environments. The biological solution is the evolution from one generation to the next one by eliminating ill-suited designs and keeping a set of near-optimal ones. Nature has a way of preventing the development of species from getting stuck at local optima by the important concept of mutation. Mutation can be defined as a pseudo-random process which alters the characteristics of the design in an unpredictable way. The altered design is again subjected to the evolutionary process which determines its survival. The main goal of our work was to adapt the power of evolution and mutation, and use it for engineering purposes like the problem of placing equal-sized modules in a rectangular grid which is described in this paper. The algorithm has been implemented in a PASCAL program, and has been tested on a wide variety of examples. Some typical performance results are discussed. The algorithm that is described in the paper is extremely fast, simple and efficient.

Acknowledgment: This research has been supported by the National Aeronautics and Space Administration under contract NASA NAG-1-613.

All communications should be addressed to the second author.

I. INTRODUCTION

The placement problem for N modules is known to be NP-complete and therefore the computation of an exact solution for more than a few modules is not feasible. Several heuristics for module placement based on iterative improvement and min-cut partitioning have been proposed in the past [1, 2]. Many of those algorithms cannot substantially overcome the problem of getting trapped at local minima since they are purely of greedy nature. Recently, a general combinatorial optimization technique called Simulated Annealing has been applied to solve the placement problem [3, 4]. It has been theoretically proved that the simulated annealing algorithm converges to an optimum solution and is able to produce near-optimum results of high quality given enough time to reach equilibrium. However, the simulated annealing algorithm has several undesirable features, most importantly, the need for a time/temperature schedule and its immense CPU time requirements [4].

The new algorithm proposed in this paper is based upon the analogy of finding an optimally suited modification of an initial placement to modifications in biological environments. The biological solution is the evolution from one generation to the next one by eliminating ill-suited designs and keeping a set of near-optimal ones. Nature has a way of preventing the development of species from getting stuck at local optima by the important concept of mutation. Mutation can be defined as a pseudo-random process which alters the characteristics of the design in an unpredictable way. The altered design is again subjected to the evolutionary process which determines its survival. The main goal of our work was to adapt the power of evolution and mutation to use it for engineering purposes like the problem of placing equal-sized modules in a rectangular grid as described in this paper.

The major aspects of this work are in:

(1) Adapting the evolution idea to the placement problem;

(2) Implementing the algorithm in a PASCAL program;

(3) Evaluating the algorithm performance.

II. THE EVOLUTION-BASED ALGORITHM

The current version of the algorithm is designed for placement of equal-sized square modules in a rectangular grid. The main idea is to show that this totally new concept is capable of solving VLSI-design problems in a very efficient way. The underlying concept, however, is not restricted in any way to the discussed application and future work will include enhanced capabilities as well as adaptions to different problems. A main design aspect was to make full use of the algorithm's inherent simplicity in order to avoid time-consuming computation. Although the current version heavily uses floating point numbers, this is only done to facilitate the code and will be replaced by purely integer computations in future versions so that a significant speed-up can still be expected. Nonetheless, the current version proves to be very fast, as will be shown later.

A basic concept used in the evolution-based algorithm is the establishment of a means of computing the *goodness* of the placement of an individual module relative to the placement of other modules. In order to simplify the explanation of this concept, we need to define the following terms:

N	=	number of modules
T(j)	=	normalized theoretically optimal placement value of module j
R(j)	=	current normalized real placement value of module j

Let us consider the placement of two modules. We assume that the nets connecting the modules are connected at the center of each module for simplicity hence, it can be concluded that an optimum interconnection is achieved if the two modules are adjacent. A sub-optimum placement can be achieved by placing the modules, so that they have a common corner. If they are aligned, but have another module in between them, the placement is even worse. Following this model, a computation window can be created, as shown in Fig. 1. The module for which the *goodness* of placement is to be evaluated is located in the center of the window. The interconnections with the nearest directly adjacent modules are assigned a weight of 100%. The interconnections to modules with a common corner are assigned a weight of 70%, which corresponds roughly to the additional wire length. The assignment of weights to modules at different relative positions with a given module at the center of a window is done for at most 24 connected modules. If the center module has more than 24 connections, they are ignored. (In special cases, it might be appropriate to expand or reduce the window size. Also if the x and y dimensions of the modules are significantly different, the window weights in the different directions may be differently assigned.)

For a module j, the theoretically optimal placement value $T(j)$ can be computed by placing the first four modules with highest connectivity with j in the positions with 100% weights, the next four ones in the positions with 70% weights, and so on until no more modules are to be connected, or the maximum of 24 modules is reached. It has to be stressed, that $T(j)$ is a theoretical optimum value for placement of a module relative to other modules. In practice, it might not be possible to achieve this placement, e.g., modules that happen to be placed at the edges of the chip only have three adjacent neighbors and optimality requirements for different modules are likely to collide. However, the values of $T(j)$ provide a means of evaluating the ranks of the modules in terms of their possible placement values and also serve as a means to evaluate the current placement.

Using the window, the real placement value $R(j)$ of a module j can be computed simply by overlaying it over the module and evaluating the connectivities to its 24 neighbors (or less at edges) and summing up the products of each connectivity and associated weights.

The placement values for the whole set are computed by summing up all individual values and dividing them by the number of modules N. The *goodness* of a module j currently at a specific place can be computed simply as $\frac{R(j)}{T(j)} * 100\%$. The theoretical values $T(j)$ can be precomputed as they are specific to the module set and connectivity, but not to the placement itself, while the real values $R(j)$ are computed for each placement.

Since a means of evaluating and comparing a placement has been established, we now describe the design of an algorithm for module placement as shown in the block-diagram in Fig. 2. The algorithm tries to maximize the placement value for each module: this is equivalent to minimize the wire length. The blocks will subsequently be described in detail:

INIT	The variables, arrays and window are initialized. The user is prompted for two process parameters. An input file containing a connectivity matrix of the modules to be placed is read. The precomputations of the theoretically optimum placement values $T(j)$ for each module j and the whole assembly are performed.
PLACE	An initial placement is performed. Since the algorithm does not depend on its quality, the modules are simply assigned to free spaces in order of occurrence.

— loop :

MUTATE	The placement is mutated with the probabilities the user has entered. This is done by pairwise exchange of randomly selected modules.
EVALUATE	The placement is evaluated by overlaying the window over each module and computing its placement value. The total placement value is computed.
JUDGE	The decision regarding whether a module is to be newly allocated is performed by comparing the normalized placement value with a random value of linear distributions. This placement value is equivalent to the survival chance of the module at its present location.
SORT	The set of modules to be newly placed are sorted in order of maximum connectivity. Modules that are in the beginning of the sorted sequence will be placed before others that appear later in the sequence.
ALLOCATE	All modules scheduled for placement are removed from their old positions. The first module is taken from the sorted sequence and tentatively placed at all available positions until the best placement is found. This is repeated for all remaining modules to be placed.

I/O The best placement so far obtained is saved and, in case of an update, the user is informed. If desired, the current total placement value is displayed.

— the loop is exited after a specified placement value, CPU time or number of steps are exceeded.

Two user-specified constants determine the probability of mutation and the percentage of modules to be exchanged. Simulations have shown that the optimal values of those constants vary among different input data sets. However, the values can be determined in a few test runs. Also, if default values are used, the final result will not be worse than a few percent of what could be achieved by fine-tuning the constants.

III. SIMULATION RESULTS

Perhaps the most startling fact about the new concept is its fast convergence. An example with 49 modules where a known 100% solution exists was found to converge within 500 iterations to the optimum solution which required approximately 10 seconds of CPU time when implemented in PASCAL running under 4.2 UNIX on a GOULD 9050 computer. The graph depicting the best value so far of the normalized placement ratio R/T (averaged over all modules) versus the number of iterations is shown in Fig. 3a. Fig. 3b shows the currently computed normalized value R/T versus the iteration number. (Due to factors mentioned earlier, the actually computed value is only 98.1% although a placement check yields that the optimal solution has in fact been found).

In order to evaluate the algorithm's performance we made up several input data sets. The first category had numbers of modules ranging from 16 to 49 and known 100% solutions. The algorithm achieved optimal placements in all cases within the first 500 iterations when appropriate constants were provided.

The second class of input data sets were generated by a special random generator which produced data sets with the following characteristics: The connectivity among the modules was not uniform, and there were clusters of modules with higher internal than external connectivity. As the input data was randomly generated, a 100% placement possibility was highly unlikely. From some approximate probabilistic calculations, we estimated the lower bounds of optimality to range between 50% and 70% of the theoretical optimum. The actual values are strongly dependent upon the total number of modules. In the case of 49 modules, the lower bound is about 50%, and the algorithm achieves about 70%. Although no upper bound can be given, it can be confidently assumed that a near-optimum placement has been found, as the clusters can be easily recognized in the computed placement (see Fig. 4). The different shapes around the module numbers reflect their affiliations to different clusters.

IV. CONCLUSIONS

The proposed concept has been shown to be extremely fast, simple to understand, easy to use and extremely efficient. It has been demonstrated in an application for the placement of equally-sized square modules. However, the concept itself is not restricted in any way to this application. On the contrary, it is so generally useful that it is suitable for many other applications in VLSI-design such as placement of arbitrarily sized modules, routing problems and the like. Moreover, applications can be found in nearly all areas of engineering where efficient heuristics are required.

We plan to perform the following enhancements in the future:

- operation on net-list input data rather than connectivity data

- enhanced variety of features like ability to place arbitrarily sized modules

- a parallel version of the algorithm (note the strong inherent parallelism of the proposed concept).

REFERENCES

[1] M. Hanan and J. M. Kurtzberg, "Placement Techniques," in *Design Automation of Digital Systems: Theory and Techniques*, ed., M. A. Breuer. Prentice-Hall, pp. 213-282, 1972.

[2] M. A. Breuer, "Min-cut Placement," *Jour. Design Automation and Fault Tolerant Computing*, vol. 1, pp. 343-382, Oct. 1977.

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, pp. 671-680, May 1983.

[4] C. Sechen and A. S. Vincentelli, "The TimberWolf Placement and Routing Package," *Proc. Custom Integrated Circuits Conf.*, pp. 522-527, May 1984.

.1	.2	.5	.2	.1
.2	.7	1	.7	.2
.5	1	▨	1	.5
.2	.7	1	.7	.2
.1	.2	.5	.2	.1

Fig.1 : Evaluation window

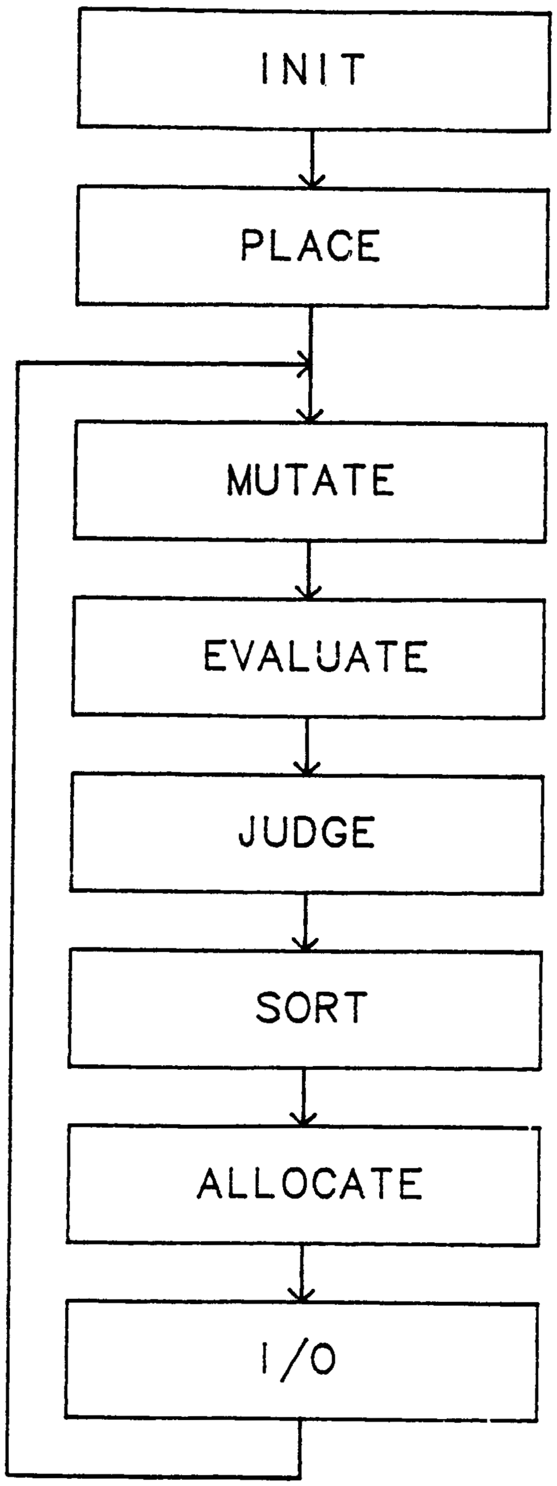

Fig.2 : Algorithm outline

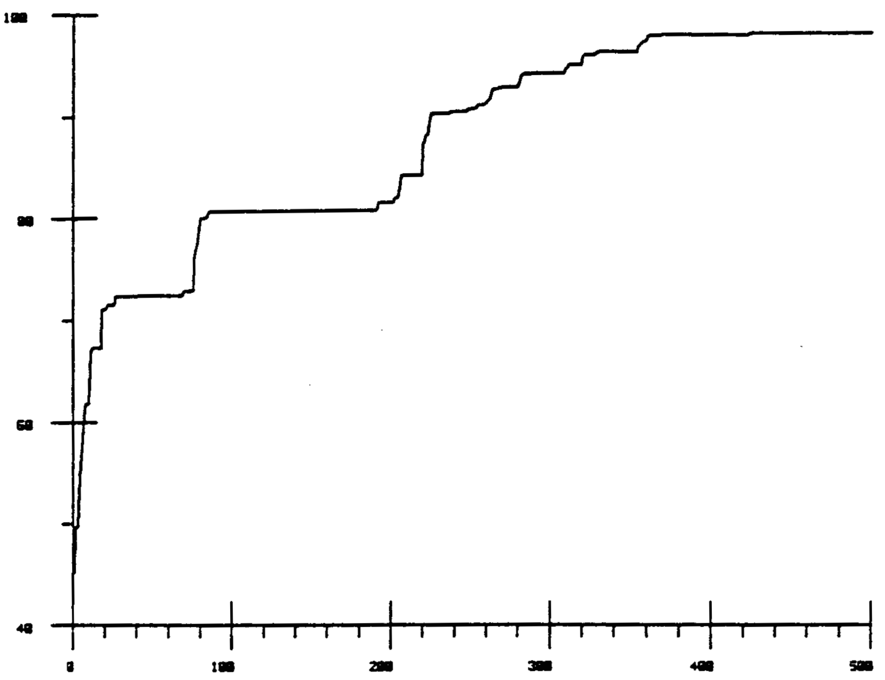

Fig.3a : Normalized best placement value versus interation number

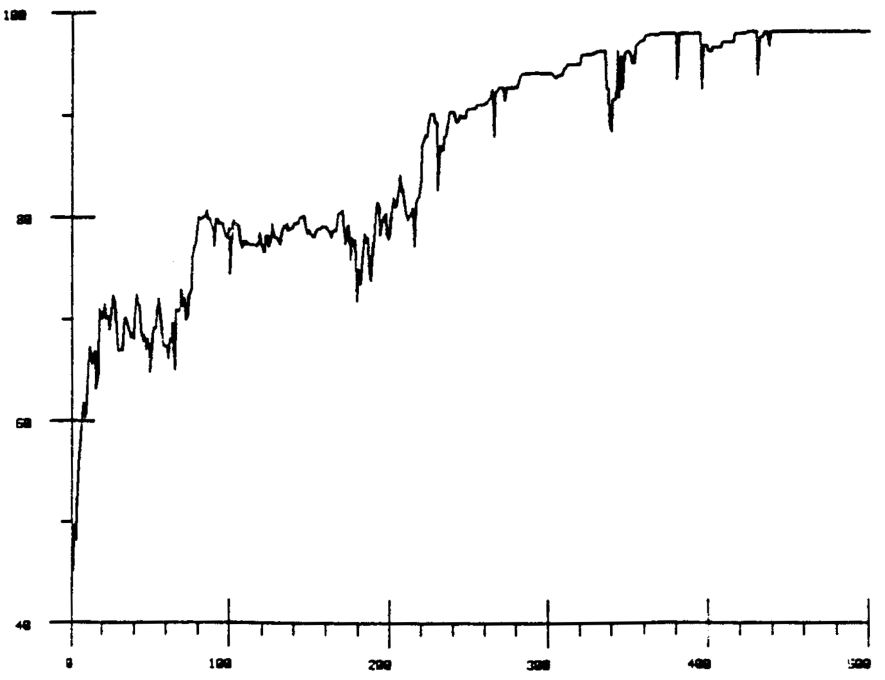

Fig.3b : Normalized current placement value versus iteration number

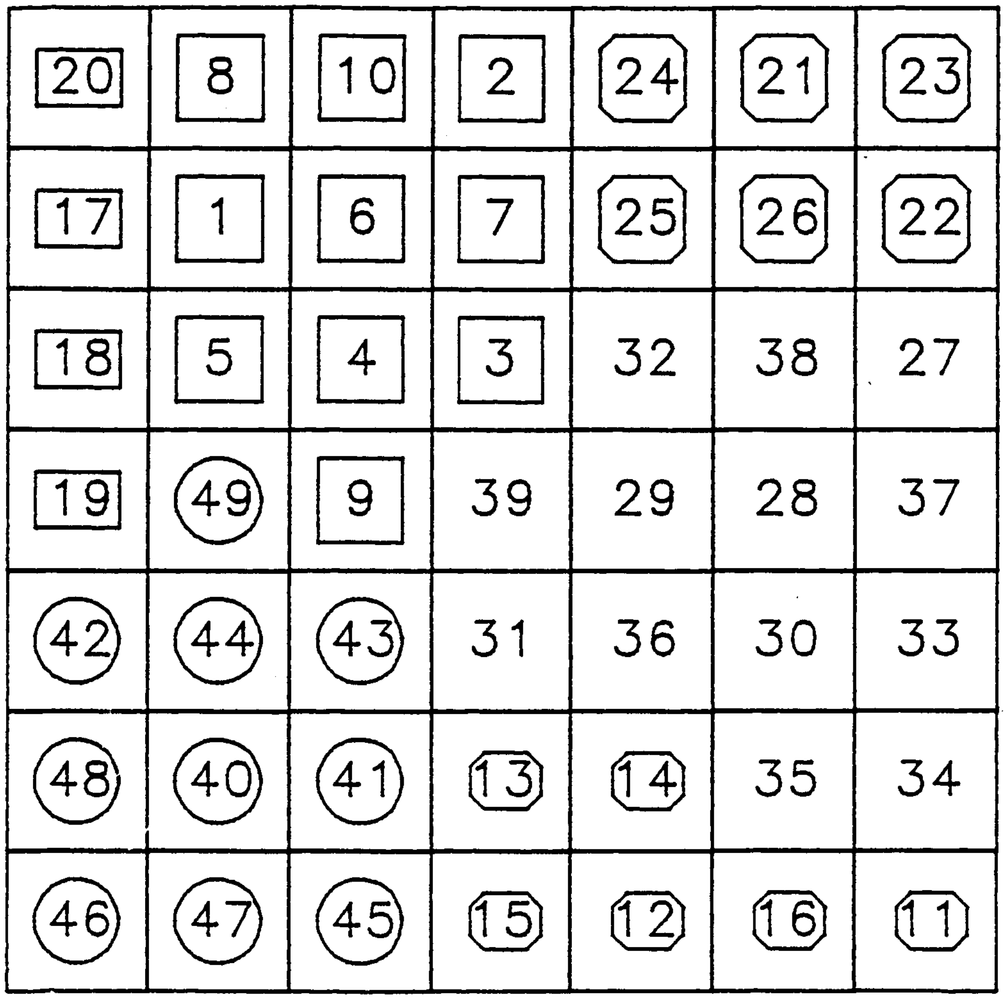

Fig.4 : Computed near-optimal placement